Spécifications détaillées du produit SimPA2

France, USA, Germany, Japan, Korea, United Kingdom, Brazil, Hungary

IMAGINE SA - 7, place des Minimes – 42300 Roanne FRANCE

Phone +33 4 77 23 60 30 - Fax +33 4 77 23 60 31

E-mail : imagine@amesim.com

www.amesim.com

Advanced Tools & Expertise in Systems Engineering
Aeronautics - Automotive - Power Hydraulics

Contributors:

Contributors	Company	Function	Email	Telephone
D. TALBI	LMS Imagine	Developer	talbi@amesim.com	+(33)477236030
S. FURIC	LMS Imagine	Development Project Manager	furic@amesim.com	+(33)477236030
D. FARGETON	LMS Imagine	Product Director	fargeton@amesim.com	
M.Najafi	INRIA	Ingénieur Expert	masoud.najafi@inria.fr	+(33)139635907
N. CUVILLIER	ALYOTECH (CRIL)	Developer	nicolas.cuvillier@alyotech.fr	+(33)130672349

Document revisions:

Version No.	Date	Author	Observations
1	November 13th 2007	E. D. TALBI	Initial revision
1	November 16th 2007	S. FURIC	
1.1	November 19th 2007	D. FARGETON	Adjustments before send to INRIA & CRIL
1.2	November 30th 2007	N. CUVILLIER	Adjustments by ALYOTECH (CRIL) See § 5.3, 6.3.7, 6.4.1, 6.4.2, 6.7.6, 6.8.2
1.3	Novembre 30th 2007	M. Najafi	Specification of developments related to Scicos
1.4	December 3d 2007	D. FARGETON	Adjustments before first official delivery
1.5	December 4th 2007	E. D. TALBI	Remove revision for official delivery
1.6	January 23rd 2008	E. D. TALBI	Update
1.7	April 2nd 2008	E. D. TALBI	Update
1.8	July 28th 2008	E. D. TALBI	Update
1.9	February 20th 2009	E. D. TALBI	Update

Checked:

Who	Company	Function	Date / Signature
S. Furic	LMS Imagine	Development Project Manager	Dec 4th 2007
S. Furic	LMS Imagine	Development Project Manager	Aug 25th 2008

Approved:

Who	Company	Function	Date / Signature
D. Fargeton	LMS Imagine	Product Director	Dec 4th 2007
D. Fargeton	LMS Imagine	Product Director	Sep 1st 2008

Table of contents

Document revisions: **3**
Table of contents **4**
Glossaire **9**
Introduction **10**
Chapitre I : Compilateur Modelica **11**
1. Développement du compilateur : Choix techniques 11
2. Vue d'ensemble du compilateur 12
2.1. Partie amont du compilateur *12*
2.1.1. Analyse lexicale et grammaticale 12
2.1.2. Analyse sémantique 13
2.1.3. Mise à plat du modèle 13
2.2. Partie avale du compilateur *13*
2.2.1. Analyse structurelle de la solvabilité du système d'équation 13
2.2.2. Manipulations formelles 14
2.2.3. Génération de code ciblé pour un simulateur donné 14
Chapitre II Réponses à la liste des exigences fonctionnelles **15**
1. Interface homme-machine 15
1.1. Initialisation des valeurs de variables et paramètres *15*
1.1.1. Besoin fonctionnel 15
1.1.2. Définition de la valeur initiale d'une variable ou paramètre dans le code Modelica 15
1.1.3. Visualiser et modifier les valeurs initiales à partir de l'IHM d'AMESim 15
1.1.4. Visualiser et modifier les valeurs initiales à partir de l'IHM de Scicos 15
1.1.5. Liste des travaux 15
1.2. Possibilité de choisir le formalisme équationnel pour construire le modèle *16*
1.2.1. Besoin fonctionnel 16
1.2.2. Développements dans AMESim 16
1.2.3. Développements dans Scicos 17
1.2.4. Liste des travaux 17
1.3. Formalisme Schémas-Blocs *17*
1.3.1. Besoin fonctionnelle 17
1.3.2. Liste des travaux 18
1.4. Formalisme Bond Graph *18*
1.4.1. Besoin fonctionnel 18
1.4.2. Choix techniques 18
1.4.3. Développements prévus 18
1.5. Formalisme Technologique *18*
1.5.1. Besoin fonctionnel 18
1.5.2. Développements dans AMESim 18
1.5.3. Développements dans Scicos 18

1.5.4. Liste des travaux 18
1.6. Possibilité de rentrer des cartographies ou abaques 19
1.6.1. Besoin fonctionnel 19
1.6.2. Choix techniques 19
1.6.3. Développements dans le compilateur Modelica 19
1.7. Possibilité de fixer et libérer les variables lors de l'initialisation pour le calcul inverse 21
1.7.1. Besoin fonctionnel 21
1.7.2. Aide à l'initialisation dans Scicos 21
1.7.3. Aide à l'initialisation dans AMESim 22
1.7.4. Liste des travaux 22
1.8. Accès aux informations sur la structure des équations, sur leur orientation et sur les conditions d'inversibilité du modèle sous l'ensemble des formalismes 22
1.8.1. Besoin fonctionnel 22
1.8.2. Accès aux informations structurelles des modèles 22
1.8.3. Liste des travaux 22
2. Vérification statique (avant simulation) 22
2.1. Vérification de l'inversibilité du modèle par analyse structurelle et fourniture des conditions d'inversibilité 22
2.1.1. Besoin fonctionnel 22
2.1.2. Choix techniques 23
2.1.3. Analyse structurelle pour l'inversion de modèles physiques 23
2.1.4. Liste des travaux 23
2.2. Détection des problèmes mal posés 23
2.2.1. Besoin fonctionnel 23
2.2.2. Gestion des messages d'erreurs dans le compilateur 24
2.2.3. Analyse structurelle de la solvabilité dans le compilateur 24
2.2.4. Liste des travaux 24
Détection des index élevés 24
2.3.1. Besoin fonctionnel 24
2.3.2. Liste des travaux 24
2.4. Vérification du réseau (cohérence des ports/connexions) 24
2.4.1. Besoin fonctionnel 24
2.4.2. Développements dans le compilateur Modelica 24
2.4.3. Liste des travaux 25
2.5. Détection des Boucles Algébriques 25
2.5.1. Besoin fonctionnel 25
2.5.2. Détection des boucles algébriques dans AMESim 25
2.5.3. Détection des boucles algébriques dans Scicos 25
2.3.4. Liste des travaux 25
3. Génération de modèle 25
3.1. Informations sur le système 25
3.1.1. Besoin fonctionnel 25
3.1.2. Génération des informations sur le modèle dans AMESim 25
3.1.3. Génération des informations sur le modèle dans Scicos 25
3.1.4. Liste des travaux 25

3.2. Génération de code temps réel embarqué à partir d'un contrôleur discret *26*
3.1.1. Besoin fonctionnel 26
3.1.4. Liste des travaux 26
4. Initialisation 26
4.1. Calcul Inverse *26*
4.1.1. Besoin fonctionnel 26
4.1.2. Résolution de l'état initial comme un problème inverse dans Scicos 26
4.1.3. Résolution de l'état initial comme un problème inverse dans AMESim 26
4.1.4. Liste des travaux 27
4.2. Résolution de l'état initial comme état stationnaire *27*
4.2.1. Besoin fonctionnel 27
4.2.2. Résolution de l'état initial comme état stationnaire dans AMESim 27
4.2.3. Résolution de l'état initial comme état stationnaire dans AMESim 27
3.2.4. Liste des travaux 27
4.3. Résolution du problème de commutation de modèles pour l'initialisation *27*
4.3.1. Besoin fonctionnel 27
4.3.2. Développements dans le compilateur Modelica 27
4.3.3. Liste des travaux 28
5. Simulation 28
5.1. Calcul direct *28*
5.1.1. Besoin fonctionnel 28
5.1.2. Simulation directe sous AMESim 28
5.1.3. Simulation directe sous Scicos 28
5.1.4. Liste des travaux 28
5.2. Résolution du problème de commutation de modèles pour la simulation dynamique *28*
5.2.1. Besoin fonctionnel 28
5.2.2. Développements dans le compilateur Modelica 28
3.2.3. Liste des travaux 28
5.3. Traitement des boucles Algébriques (DAE) *28*
5.3.1. Besoin fonctionnel 28
5.3.2. Traitement des boucles algébriques dans AMESim et Scicos 28
5.3.3. Liste des travaux 28
5.4. Traitement des systèmes Raides (DAE et ODE) *29*
5.4.1. Besoin fonctionnel 29
5.4.2. Traitement des systèmes raides dans AMESim et Scicos 29
5.4.3. Liste des travaux 29
5.5. Traitement des systèmes hybrides *29*
5.5.1. Besoin fonctionnel 29
5.5.2. Spécification du nouveau modèle hybride 29
5.5.3. Développements dans le compilateur Modelica 29
5.5.4. Liste des travaux 29
5.6. Calcul inverse automatisé en cours de simulation *30*
5.6.1. Besoin fonctionnel 30
5.6.2. Résolution du problème inverse dynamique 30
5.6.3. Liste des travaux 30

5.7. Couplage de modèles 1D et 3D *30*
5.7.1. Besoin fonctionnel 30
5.7.2. Résolution des problèmes de couplage 1D et 3D en utilisant Modelica 30
5.7.3. Liste des travaux 30
6. Outils de diagnostic 30
6.1. Informations sur la structure des équations effectivement résolues *30*
6.1.1. Besoin fonctionnel 30
6.1.2. Développements dans AMESim et Scicos 30
6.1.3. Liste des travaux 30
6.2. Accès aux informations sur la structure des équations, leur orientation et les conditions d'inversibilité du modèle (version textuelle) *30*
6.2.1. Besoin fonctionnel 30
6.2.2. Développements dans le module d'analyse structurelle pour l'inversion de modèles 31
6.2.3. Liste des travaux 31
6.3. Messages d'erreurs en cours de simulation *31*
6.3.1. Besoin fonctionnel 31
6.3.2. Affichage des messages d'erreurs en cours de simulation dans AMESim 31
6.3.3. Affichage des messages d'erreurs en cours de simulation dans Scicos 31
6.3.4. Liste des travaux 31
7. Environnement informatique 31
7.1. Plateformes informatiques supportés *31*
7.1.1. Besoin fonctionnel 31
7.1.2. Environnements informatiques supportés 31
7.1.3. Liste des travaux 31
7.2. Mise à disposition de la documentation utilisateur *31*
7.2.1. Besoin fonctionnel 31
7.2.2. Liste des travaux 31
8. Les performances 31
8.1. Besoin fonctionnel 31
8.2. Développements dans le compilateur Modelica pour AMESim *32*
8.2. Liste des travaux 32
9. La portabilité des modèles entre AMESim et Scicos 32
9.1. Besoin fonctionnel *32*
9.2. Compatibilité des modèles entre AMESim et Scicos *32*
9.2. Liste des travaux *32*
10. La documentation 32
10.1. Besoin fonctionnel *32*
10.2. Documentation du compilateur Modelica *32*
10.3. Liste des travaux *32*
11. Autres besoins fonctionnels 33
11.1. Besoins liés au retour d'expérience sur SimPA *33*
11.1.1. Besoin fonctionnel 33
11.1.2. Développements dans le compilateur Modelica 33
11.1.3. Liste des travaux 33
11.2. Besoins liés aux premiers tests sur SimPA2 ..... *33*
11.2.1. Besoin fonctionnel 33

11.2.2. Développements dans le compilateur Modelica 33
11.2.3. Liste des travaux 33
11.3. Besoins liés au développement de librairies 33
11.3.1. Besoin fonctionnel 33
11.3.2. Développement dans le compilateur Modelica 34
11.3.3. Liste des travaux 34
Chapitre III : Limitations et récapitulatif des réalisations 35
1. Principales limitations du compilateur Modelica 35
2. Tableau récapitulatif des fournitures du projet 36
Références 38

Glossaire

Caml ou Ocaml: langage de programmation développé et distribué par l'INRIA depuis 1985 (consulter le site http://caml.inria.fr pour plus de détails).

Classe: fait référence au concept de « classe » du langage Modelica. Une classe fournit la description des objets qui lui appartiennent (cf. [7]).

Equation: fait référence au concept d'équation dans le langage Modelica (cf. [5] Chapter 8).

Expression: fait référence à la notion d'expression telle que décrite dans la grammaire du langage Modelica (cf. [5] Appendix B).

Graphe: est un ensemble de sommets (appelés aussi nœuds) reliés par des arcs (cas d'un graphe orienté), ou des arêtes (cas d'un graphe non orienté).

IHM (Interface Homme-Machine): dénote l'ensemble des moyens mis en place pour assurer la communication entre l'homme et la machine.

Instance: un objet particulier en tant qu'élément d'une classe donnée.

Instanciation: procédure permettant d'obtenir un objet avec un paramétrage donné à partir d'une classe du langage Modelica.

Modelica: langage pour la modélisation des systèmes physiques, développé par l'association Modelica (consulter le site http://www.Modelica.org/ pour plus de détails).

Modelicac : Partie avale du compilateur Modelica dont la version initiale a été développée au cours du projet SimPA, puis améliorée cours du projet SimPA2.

Objet: entité du langage représentant :
- les modèles manipulés par l'utilisateur,
- les données structurées (enregistrements).

Translator : Partie amont du compilateur Modelica développé entièrement dans le cadre du projet SimPA2.

Type: cf. [5] Chapter 6, pour une définition de ce concept.

Variabilité: concept du langage Modelica, permettant de contraindre l'ensemble des valeurs pouvant être prises au cours du temps, par une expression.

XML: « eXtensible Markup Language » est un langage utilisé principalement pour le stockage et l'échange de données.

Introduction

SimPA2 (Simulation pour le Procédé et l'Automatique) est un projet labellisé ANR/RNTL en 2005, dont l'objectif principal est le développement d'un compilateur Modelica open source et son intégration au produit open source Scilab/Scicos et au produit commercial AMESim. Ce projet fait suite au projet SimPA qui a permis d'obtenir une première version de compilateur Modelica open source, qui sera utilisée comme point de départ pour l'implémentation du compilateur Modelica de SimPA2.

Ce document est la spécification détaillée du projet SimPA2 requise dans le cadre du lot 0 du sous-projet 1 (cf. l'annexe technique du projet [2]). Il est divisé en trois parties :

- La première partie donne une description du point de vue fonctionnel et architecture logicielle du compilateur Modelica,
- La seconde partie est une réponse à la liste des exigences fonctionnelles [3] établie par les partenaires industriels (cf. l'annexe technique du projet [2]),
- La dernière partie est un récapitulatif des principales limitations du compilateur, ainsi que le planning des livrables du projet.

Dans ce qui suit, nous appellerons compilateur de SimPA l'ancienne version du compilateur développée au cours du projet SimPA (appelée aussi Modelicac). La nouvelle version (celle du projet SimPA2, dont ce document est la spécification détaillée) sera dénommée « compilateur Modelica ».

Chapitre I : Compilateur Modelica

1. Développement du compilateur : Choix techniques

Le projet SimPA2 s'appuie sur les résultats du projet SimPA qui a permis de mettre au point une première version de compilateur Modelica. Celle-ci couvre un ensemble très restreint du langage Modelica et ne permet pas de traiter la complexité des cas industriels. L'objectif principal du projet SimPA2 est de développer une version améliorée du compilateur pour traiter au mieux les exigences des partenaires industriels du projet.

En partant du compilateur SimPA, une première possibilité consiste à réutiliser une partie du compilateur OpenModelica, qui est un produit « open source » développé par le laboratoire PELAB de l'université de Linköping (Suède). Ce choix d'utiliser Open-Modelica a été rejeté pour diverses raisons, parmi lesquelles on peut citer :

- la version d'OpenModelica disponible au début du projet SimPA2 (en début 2006) est très limitée : aucun support des équations conditionnelles, les tableaux définis en compréhension, les conversions implicites de types…,
- la sémantique du langage Modelica reste assez ambiguë concernant plusieurs aspects importants pour le projet SimPA2, comme les aspects discret ou l'utilisation des fonctions externes. Ce qui implique la nécessité d'une refonte complète de toute la partie analyse sémantique d'OpenModelica.
- les discussions au sein de l'association Modelica laisse présager un grand nombre de changements dans les versions futures des spécifications du langage, ce qui peut remettre en cause une grande partie des développements d'OpenModelica.
- OpenModelica n'effectue qu'une mise à plat très partielle des modèles, ce qui rend son adaptation au format accepté par modelicac très couteux.
- la détection des erreurs de type est très insuffisante dans OpenModelica.
- OpenModelica est un compilateur créé pour des objectifs plus académiques et de recherche, et semble assez inadapté à une utilisation industrielle.

Pour toutes les raisons précédentes, nous avons opté pour une autre approche qui consiste à utiliser une évolution de Modelicac comme partie avale du compilateur, et développer une partie amont qui s'occupera de traiter les éléments du langage Modelica avant de les transmettre à modelicac sous une forme qu'il accepte. Cette seconde approche offre aussi plusieurs avantages :

- la possibilité de s'interfacer plus facilement avec les différents outils utilisés et développés dans le cadre du projet,
- plus de liberté pour faire évoluer le langage Modelica,
- la liberté de faire les choix techniques adaptés,
- plus de facilité pour modifier ou réadapter le code en fonction de l'évolution des besoins du projet.

Pour développer la partie amont du compilateur nous avons opté pour l'utilisation du langage de programmation OCaml développé et distribué par l'INRIA depuis 1985. Le choix de ce langage a été fait pour plusieurs raisons :

- Ce langage fait partie de la famille des langages « ML » (contraction de Meta Language) qui sont bien adaptées à l'écriture de compilateurs,
- Ce langage est réputé pour être performant et très sûr,
- Ce langage est un projet « open source », ce qui offre plusieurs avantages en termes de qualité, maintenance et pérennité des produits développés.

2. Vue d'ensemble du compilateur

Le compilateur Modelica permet d'analyser du code Modelica, de l'interpréter et de générer en sortie du code à destination d'un outil de simulation de systèmes physiques. Il est composé de deux parties (voir la figure 1) :

- une partie amont (appelée aussi « translator »),
- une partie avale (appelée aussi « modelicac[1] »).

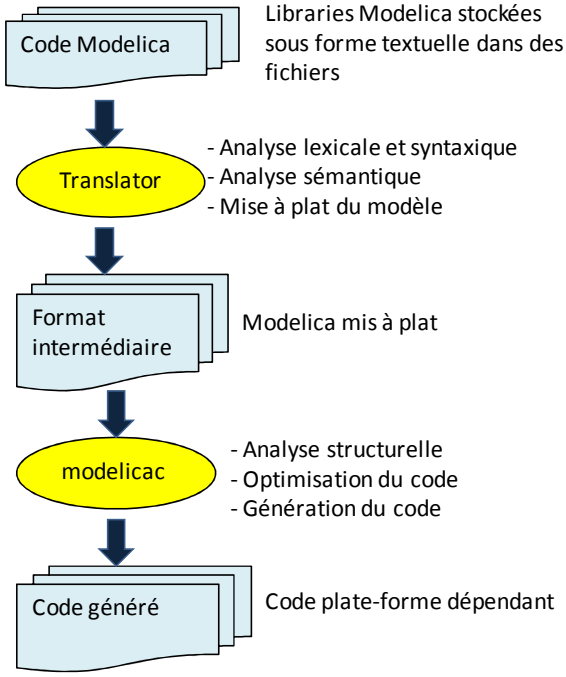

Figure 1 – Schéma général du compilateur Modelica

La partie amont extrait le sens du code Modelica fourni en entrée et le traduit sous une forme intermédiaire, qui est transmise à la partie avale pour être transformée en une forme exploitable par un solveur numérique. Dans ce qui suit nous aborderons de manière générale les traitements effectués dans chacune des deux parties : amont et avale. Ensuite, tout au long du document nous donnerons plus de détails sur les développements réalisés pour répondre aux exigences des partenaires industriels du projet.

2.1. Partie amont du compilateur

Cette partie effectue les traitements suivants sur le code Modelica reçu en entrée :

- Analyse lexicale et grammaticale,
- Analyse sémantique,
- Mise à plat du modèle.

Ces trois étapes sont décrites dans ce qui suit :

2.1.1. Analyse lexicale et grammaticale

Au cours de l'analyse lexicale le flux de caractères (provenant du code source Modelica) est transformé en une suite de lexèmes. Chaque lexème représente une entité élémentaire du langage Modelica : mot-clé, opérateur, identifiant, commentaire… Si des caractères non autorisés sont détectés, une erreur lexicale est affichée.

[1] La partie avale du compilateur est une version améliorée du compilateur de SimPA.

Les lexèmes sont ensuite regroupés en phrases par l'analyseur grammatical qui détecte les phrases mal formées et renvoi des messages d'erreurs de syntaxes éventuels. Si cette phase est passée avec succès, un arbre de syntaxe abstraite est produit. Si une erreur se produit au cours de cette étape d'analyse, une erreur de syntaxe est affichée.

2.1.2. Analyse sémantique

Au cours de cette étape, le compilateur tente de donner des sens aux phrases (grammaticalement correctes) obtenues suite à l'étape précédente (analyse lexicale et grammaticale). Si une phrase n'a pas de sens ou introduit une construction du langage non supportée par le compilateur, une erreur de sémantique est affichée.

2.1.3. Mise à plat du modèle

Au cours de cette étape, plusieurs traitements sont effectués afin de réécrire le système à traiter sous une forme adaptée aux manipulations symboliques. Parmi les traitements qui sont effectués :

- mise à plat des équations et variables multidimensionnelles,
- transformation des équations conditionnelles en expressions conditionnelles,
- expansion des opérateurs et fonctions internes au langage,
- génération des équations de connexion,
- substitution des paramètres fixés (i.e. attribut « fixed » égal à « true ») par leurs valeurs dans les équations,
- génération des équations initiales,
- dérivation formelle d'expressions…

Suite à cette étape plusieurs codes sont générés :

- Un code intermédiaire pour la résolution du problème d'initialisation,
- Un code intermédiaire pour la résolution du problème dynamique,
- Une description XML des dépendances entre variables.

Nous présenterons par la suite une description détaillée de l'utilisation de chacun des codes précédents.

2.2. Partie avale du compilateur

La partie avale, qui reçoit en entrée un système d'équations décrit à l'aide du format intermédiaire produit par « translator », va opérer un ensemble de transformations visant à passer d'une forme dite « déclarative » à une forme dite « procédurale ».

La forme déclarative est celle du code Modelica de départ, elle permet d'exprimer les problèmes avec un très haut niveau d'abstraction : on ne décrit pas comment résoudre mais seulement les contraintes devant être vérifiées par le résultat. Cette forme se prête bien aux spécifications d'un problème mais n'est pas exploitable directement par les codes de calcul.

La forme procédurale décrit les séquences de calcul menant à l'obtention d'un résultat numérique à l'intérieur d'une boucle de recherche de point fixe (appel d'un solveur numérique). Cette forme est relativement éloignée des spécifications de départ et ne retient que ce qui est nécessaire au code de calcul.

Les traitements effectués par la partie avale du compilateur sont principalement :

- Analyse structurelle de la solvabilité du système d'équation,
- Manipulations formelles,
- Génération de code ciblé pour un simulateur donné.

2.2.1. Analyse structurelle de la solvabilité du système d'équation

Dans cette étape, le compilateur analyse la structure du système d'équations pour détecter certains cas de problèmes insolubles (i.e. systèmes n'admettant pas de solutions). L'analyse effectuée consiste à :

- Vérifier si le nombre de variables inconnues est égal au nombre d'équations,
- Analyser le graphe de dépendances entre équations et variables, en utilisant un algorithme de couplage dans un graphe biparti.

2.2.2. Manipulations formelles

Le noyau de manipulation formelle fait essentiellement deux choses :

- Il réduit le système d'équations (par application de règles de réécriture),
- Il détermine une séquence de calcul permettant de mettre à jour la valeur des inconnues dans le code généré (factorisation d'expressions communes et calcul des dépendances).

La représentation interne utilisée en entrée comprend l'ensemble des paramètres « non fixés » et variables du modèle à analyser, les valeurs initiales imposées ainsi que les équations. Cette représentation est exploitée pour construire deux systèmes : continu et discret.

Le système continu est décomposé en sous-systèmes autonomes par un module d'analyse de graphe d'incidence. Ces sous systèmes sont réduits par manipulations symboliques, en résolvant les parties linéaires et en réduisant au mieux les parties non-linéaires (par des méthodes d'inversion et de substitution symboliques). Le système simplifié obtenu à la fin, est constitué de sous-systèmes non-réductibles qui devront être traités numériquement. Le résultat comprendra éventuellement les paramètres dont on ne connaît pas la valeur.

Les contraintes du système discret sont ordonnancées statiquement par une méthode basée sur un algorithme issu du compilateur Scicos, fourni par l'INRIA. Les équations considérées ici seront « explicites », au sens où on va exiger que le membre de gauche soit la variable par rapport à laquelle on va résoudre l'équation en question.

2.2.3. Génération de code ciblé pour un simulateur donné

La génération de code permet aux applications clientes de récupérer de l'information structurelle sur les modèles à analyser ou du code C compilable dont l'exécution dans un environnement de simulation permet d'obtenir l'état initial et les courbes d'évolution temporelles d'un système dynamique hybride.

Les modules de génération de code sont spécifiques aux environnements cibles, car ceux-ci n'ont pas les mêmes exigences. Deux cibles pour la génération de code ont été définies :

- Scicos
- AMESim

La génération vers la première cible consiste à produire un fichier C contenant le code d'un bloc Scicos comportant des sections de calcul de résidus, de spécification de valeurs initiales, de détection de traversées de surface, de calcul de coefficients de matrice Jacobienne… Pour les besoins d'analyse, un fichier XML donnant la structure du système d'équations pourra être généré.

La génération vers AMESim consiste à produire deux fichiers :

- Un fichier C décrivant le code du sous-modèle AMESim correspondant au modèle Modelica compilé
- Un fichier XML décrivant l'interface du sous-modèle.

Chapitre II Réponses à la liste des exigences fonctionnelles

1. Interface homme-machine

1.1. Initialisation des valeurs de variables et paramètres

1.1.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de :
- définir la valeur initiale d'une variable ou paramètre dans le code Modelica,
- visualiser et modifier cette valeur à partir de l'IHM.

1.1.2. Définition de la valeur initiale d'une variable ou paramètre dans le code Modelica

La définition de la valeur initiale d'une variable ou un paramètre est supporté par le compilateur SimPA (aucun développement nécessaire) en utilisant l'attribut « start » et les équations de déclaration, comme dans les deux exemples suivants :
- Définition d'une variable x de type Real, avec une valeur initiale :
```
Real x(start = 8.0) ;
```
- Définition d'un paramètre y de type Real, avec une valeur initiale modifiable :
```
parameter Real y(fixed = false) = 2.0 ;
```

Le compilateur est amélioré pour supporter l'utilisation de l'attribut « fixed » permettant de fixer une fois pour toute la valeur d'un paramètre (le paramètre n'est pas modifiable dans le code « C » final généré), comme dans l'exemple :
- Définition d'un paramètre z de type Real, avec une valeur initiale non modifiable[2] :
```
parameter Real z = 3.0;
```

1.1.3. Visualiser et modifier les valeurs initiales à partir de l'IHM d'AMESim

Dans l'environnement AMESim, il est possible de visualiser et modifier les valeurs initiales des variables et paramètres du modèle en utilisant l'interface homme-machine (IHM) standard d'AMESim en mode paramètres. Pour améliorer la visualisation des valeurs initiales, celles-ci doivent être affichées sous une forme arborescente. Cette fonctionnalité nécessite une amélioration de la génération de code pour AMESim (dans « modelicac »).

1.1.4. Visualiser et modifier les valeurs initiales à partir de l'IHM de Scicos

Dans l'environnement Scicos, les valeurs initiales des variables et des paramètres du modèle peuvent être visualisées et modifiées en utilisant l'interface homme-machine (IHM) standard de Scicos, en cliquant sur les blocs.

1.1.5. Liste des travaux

Fonctionnalité	Responsable	Produit
Support de l'attribut « fixed »	LMS-Imagine	Compilateur Modelica
Amélioration de la visualisation des valeurs des paramètres et variables dans AMESim	LMS-Imagine	Compilateur Modelica pour AMESim

[2] L'attribut « fixed » est par défaut égale à « true » pour les paramètres.

1.2. Possibilité de choisir le formalisme équationnel pour construire le modèle

1.2.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de construire des modèles en utilisant le langage Modelica.

1.2.2. Développements dans AMESim

A partir de la version Rev7 d'AMESim (mai 2007), l'utilisateur a la possibilité d'importer et simuler dans
La version Rev7A d'AMESim (sortie en mai 2007) inclut un assistant d'import Modelica. Cet assistant demande à l'utilisateur d'indiquer :

- les chemins vers les sources Modelica à compiler
- le nom de la classe à instancier
- le nom de l'instance à générer
- le répertoire qui sera utilisé pour stocker le code généré.

Suite à cette étape de configuration, l'assistant appelle le compilateur Modelica avec les arguments appropriés, et génère (si aucune erreur ne se produit) le code C et XML permettant de décrire un sous-modèle AMESim.

L'assistant peut générer deux types de sous-modèles AMESim :

- un sous-modèle autonome (sans ports)
- un sous-modèle connectable à des sous-modèles issus de la compilation de code Modelica ou de la bibliothèque standard AMESim (cf. figure 2).

Concernant le comportement des sous modèles deux options sont possibles :

- le sous-modèle expose toutes ses variables internes à l'utilisateur (mode par défaut).
- le sous-modèle ne transmet que ses variables de sortie (mode « boîte noire »).

Dans cette première version de l'import Modelica, les variables et équations discrètes ne sont pas supportées. Pour plus de détails sur l'utilisation de l'assistant d'import, un manuel d'utilisateur est mis à disposition avec l'aide en ligne d'AMESim.

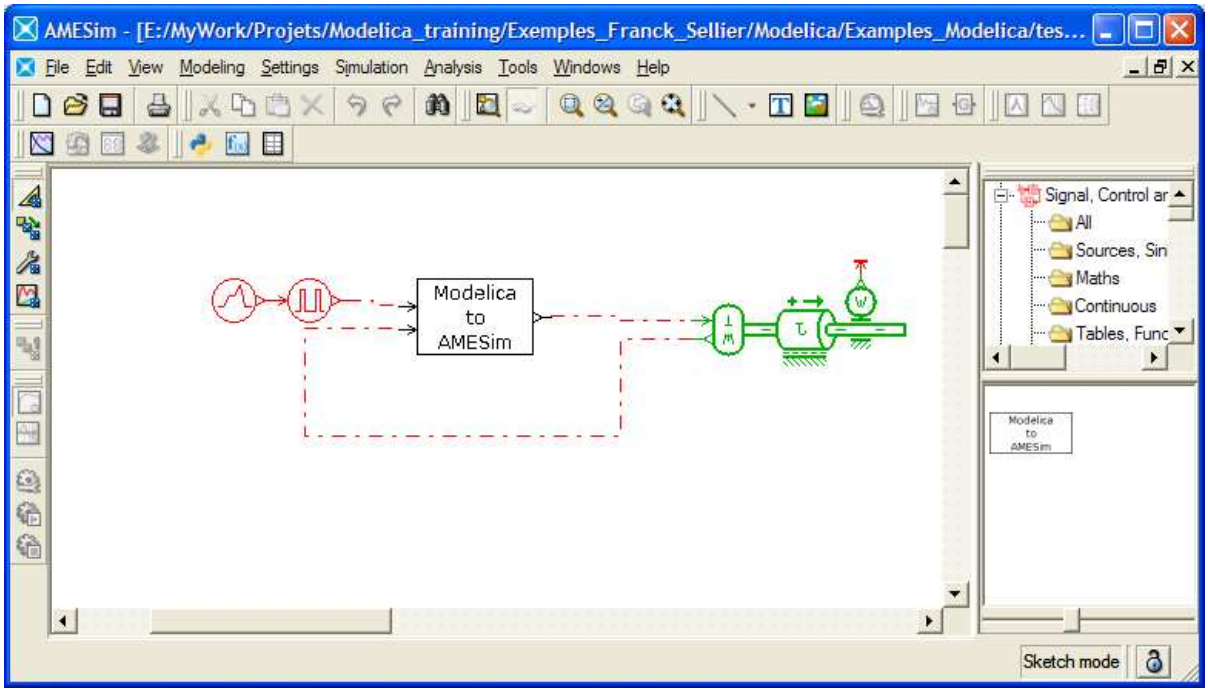

Figure 2 – *Sous-modèle AMESim constitué d'un composant Modelica connecté à des composants AMESim classiques.*

La version Rev8A d'AMESim (mai 2008) apportera plusieurs améliorations :

- Conversion des unités Modelica en unités AMESim,
- Affichage des unités pour les variables et les paramètres,
- Plusieurs améliorations dans l'interface graphique de l'assistant d'import Modelica,
- Possibilité de Modifier et recompiler un modèle Modelica, directement à partir du mode sketch d'AMESim.

1.2.3. Développements dans Scicos

Pour pouvoir utiliser les modèles Modelica dans Scicos, chaque Modèle est associé à un fichier interface de Scicos « .sci ». Ce fichier définit les entrées/sorties implicites/explicites et la forme graphique du bloc dans Scicos. La figure 3 montre le modèle d'un système de contrôle de moteur (développé par l'IFP) dans Scicos. Dans les diagrammes Scicos, l'utilisation simultanée de blocs Scicos standard et de blocs Modelica est autorisée.

Il est également possible d'utiliser des modèles Modelica dans les diagrammes Scicos sans avoir à fournir un fichier d'interface, en utilisant une interface graphique générique pour les modèles Modelica (bloc « MBlock »).

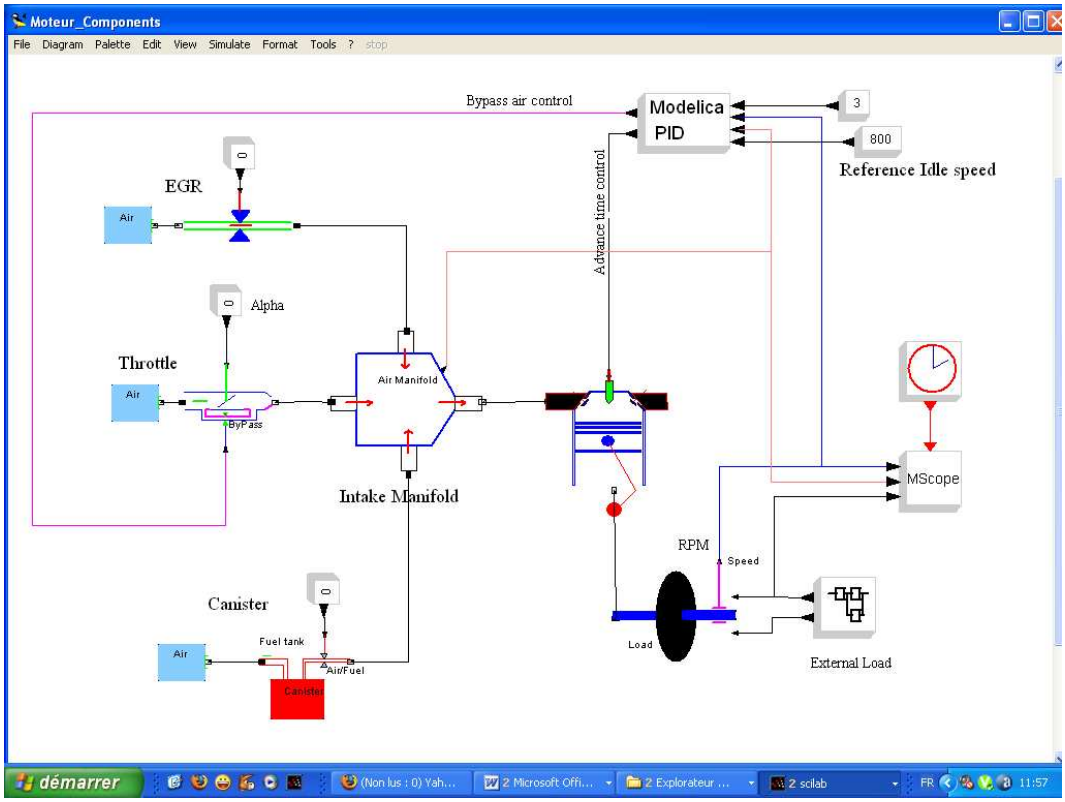

Figure 3 – *Schéma d'utilisation des composants Modelica dans Scicos.*

1.2.4. Liste des travaux

Fonctionnalité	Responsable	Produit
Assistant d'import Modelica dans AMESim	LMS-Imagine	AMESim

1.3. Formalisme Schémas-Blocs

1.3.1. Besoin fonctionnelle

Offrir dans AMESim et Scicos la possibilité de construire des modèles en utilisant le formalisme Schémas-Blocs (causal).

1.3.2. Liste des travaux

Ce besoin est entièrement couvert par les versions standards d'AMESim et Scicos (aucun développement prévu).

1.4. Formalisme Bond Graph

1.4.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de construire des modèles en utilisant le formalisme Bond-Graph.

1.4.2. Choix techniques

Au début du projet SimPA 2, les partenaires INSA et Imagine ont travaillé ensemble pour déterminer la compatibilité éventuelle des formalismes « Bond graph » et Modelica. Les conclusions de ce travail sont les suivantes :

- le formalisme « Bond graph » permet de caractériser certaines propriétés d'un modèle physique (lignes de puissance entre autres) impossibles à caractériser en Modelica (où le couplage effort-flux n'est pas explicite)
- Modelica est plus large que le formalisme « Bond graph » car il comprend les aspects discrets absents de ce dernier
- le formalisme « Bond graph » permet de spécifier un programme Modelica dans lequel certaines propriétés sont vérifiées *a priori* (comme par exemple l'inversibilité structurelle)
- il n'est par contre généralement pas possible de passer d'un programme Modelica à une représentation de type « Bond graph » (car on ne sait pas quelles variables de types effort et flux sont couplées, certains programmes mêmes purement continus ne sont pas traduisibles car ils possèdent des états liés, etc.)

Au final, il a donc été décidé de traiter le formalisme « Bond graph » en amont de la chaîne de compilation Modelica.

1.4.3. Développements prévus

Aucun développement prévu. Le logiciel MS1, utilisé par le partenaire industriel PSA, permet de générer du code Modelica à partir d'une représentation « Bond Graph ».

1.5. Formalisme Technologique

1.5.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de construire des modèles en utilisant le formalisme Technologique (assemblage de composants).

1.5.2. Développements dans AMESim

Parallèlement à la version Rev8A d'AMESim (prévue en mai 2008), une version dédiée à des projets de recherche offre la possibilité de construire un modèle Modelica par assemblage graphique de composants (sans passer par la version textuelle). L'utilisateur peut éditer un sous-modèle Modelica directement sous AMESim à l'aide de l'interface graphique. Le mélange de composants Modelica et AMESim ne sera pas autorisé, la communication entre les mondes Modelica et AMESim se réalisera à travers des ports (physiques ou signaux). La modélisation se fera donc en deux étapes :

- Construction d'un sous-modèle AMESim par assemblage de composants Modelica
- Incorporation du résultat à un modèle (circuit) AMESim et utilisation traditionnelle d'AMESim.

1.5.3. Développements dans Scicos

La construction des diagrammes avec l'assemblage des composants est possible dans Scicos. Des exemples thermo-hydrauliques et électriques sont mis à disposition dans le répertoire « demos » de Scicos.

1.5.4. Liste des travaux

Fonctionnalité	Responsable	Produit

Assistant d'assemblage Modelica dans AMESim	LMS-Imagine	AMESim

1.6. Possibilité de rentrer des cartographies ou abaques

1.6.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de définir et d'utiliser des cartographies et abaques en utilisant le langage Modelica.

1.6.2. Choix techniques

L'utilisation des cartographies et abaques à partir du langage Modelica est possible via l'une des deux options suivantes :

- Utiliser des tableaux et des algorithmes,
- Utiliser des tableaux et des fonctions externes.

La première option a été rejetée car la sémantique des algorithmes est assez ambigüe dans la version courante du langage Modelica. Nous avons donc opté pour l'utilisation des tableaux et des fonctions externes. Ceci dit, en pratique l'utilisation des fonctions externes soulève aussi plusieurs problèmes qui font toujours l'objet de discussions et d'études au sein de l'association Modelica. On peut citer en particulier, les problèmes liés à l'utilisation de fonctions impures, et la représentation des discontinuités cachées dans les fonctions externes.

1.6.3. Développements dans le compilateur Modelica

Le compilateur Modelica est amélioré pour supporter

- les tableaux de dimensions paramétriques,
- les tableaux définis en compréhension,
- les opérations sur les tableaux (et opérateurs prédéfinis)
- les fonctions externes,

1.6.3.1. Les tableaux de dimension paramétrique

Dans la version SimPA du compilateur, les dimensions des tableaux étaient fixées en utilisant des constantes numériques, comme dans l'exemple :

```
Real X[5] ;
```

Cette restriction limitait considérablement les possibilités de réutilisation des modèles construits, car la dimension d'un tableau ne pouvait être modifiée sans retoucher le code source des modèles. Ce qui impliquait la nécessité de construire autant de modèles différents qu'il y a de dimensions de tableaux. Le compilateur SimPA2, lève la restriction précédente, en offrant la possibilité d'utiliser des tableaux de dimension paramétrique, comme dans l'exemple :

```
parameter Integer n = 5;
Real X[n] ;
```

1.6.3.2. Les tableaux définis en compréhension

La définition en compréhension des tableaux apporte beaucoup de facilité à l'écriture et la lisibilité des tableaux. Par exemple l'expression :

```
{ i^2 + j^2 for i in 1:n, j in 1:m }
```

définie un tableau de dimensions n×m, dont les éléments sont obtenus en faisant la somme des carrés des indices.

1.6.3.3. Les opérations sur les tableaux

Les opérations et opérateurs développés pour les calculs sur les tableaux sont (voir la spécification du langage [5] pour plus de détails) :

- les opérations arithmétiques de base : addition, multiplication, division, soustraction, exponentiation, ces opérations incluent les opérations matricielles, ainsi que les opérations entre scalaires et tableaux.
- les opérateurs prédéfinis : « ndims », « size », « scalar », « identity », « diagonal », « zeros », « ones », « fill », « max », « min », « sum », « product ».
- vectorisation des appels de fonctions scalaires,
- indexation de tableaux avec des vecteurs d'entiers, et les expressions « : » et « end »,
- Accès aux champs d'un tableau en utilisant l'operateur « slice ».

1.6.3.4. Les fonctions externes

Plusieurs améliorations sont apportées aux fonctions externes :

- Simplification de l'utilisation des fonctions externes : Les fonctions externes étaient très difficiles à utiliser avec le compilateur SimPA, car il fallait précompiler une par une chacune des fonctions externes et les stocker dans des fichiers avant de les utiliser. Le compilateur SimPA2 simplifie ce schéma d'utilisation en détectant automatiquement toutes les fonctions externes utilisées.
- Le compilateur SimPA ne supportait que les fonctions externes « C » qui n'ont que des arguments scalaires de type « double » et qui n'ont qu'une seule sortie scalaire de type « double ». Le compilateur SimPA2 étend l'utilisation des fonctions externes aux arguments de type : chaîne de caractères, entiers et tableau de « double ».

Voici un exemple de définition (côté Modelica) et d'utilisation d'une fonction externe :

```
function F "la fonction côté Modelica"
  input Integer i;
  input String s;
  input Real x[:, :];
  output Real y;
external;
end F;

model M "le modèle"
  Real x[2, 2];
  Real y;
equation
  x = {{2.5, 3.8}, {7.2, 9.1}};
  F(42, "hello", x) = y;
end M;
```

La fonction doit être déclarée comme ceci en C :

```
double F(int i, char *s, double *x, int dim1, int dim2)
{
   int j, k;
   double sum;

   for(j = 0; j < dim1; ++j)
     for(k = 0; k < dim1; ++k)
       sum += x[i * dim1 + j];

   return sum;
}
```

1.6.3.5. Types prédéfinis « String » et « Integer »

Pour faciliter l'utilisation des fonctions externes avec des paramètres et constantes de type « String » ou « Integer », un développement supplémentaire est réalisé pour supporter l'utilisation de ces deux types.

1.6.3.6. Liste des travaux

Fonctionnalité	Responsable	Produit
Tableaux de dimensions paramétriques	LMS-Imagine	Compilateur Modelica
Tableaux définis en compréhension	LMS-Imagine	Compilateur Modelica
Opérations sur les tableaux	LMS-Imagine	Compilateur Modelica

Fonctions externes	LMS-Imagine	Compilateur Modelica
Support des paramètres et constantes de type « String » et « Integer »	LMS-Imagine	Compilateur Modelica

1.7. Possibilité de fixer et libérer les variables lors de l'initialisation pour le calcul inverse

1.7.1. Besoin fonctionnel

Offrir dans AMESim et Scicos une aide à la résolution du problème d'initialisation en intervenant sur :

- les valeurs initiales des variables (et dérivées) et des paramètres,
- le choix des variables (et dérivées) et des paramètres qui sont calculés à la résolution des contraintes initiales,
- l'aide au choix des variables et paramètres en détectant les systèmes structurellement sous-contraints ou sur-contraints.

1.7.2. Aide à l'initialisation dans Scicos

La figure 4 ci-dessous décrit les différentes étapes pour la résolution du problème d'initialisation et la génération du code de simulation dynamique final.

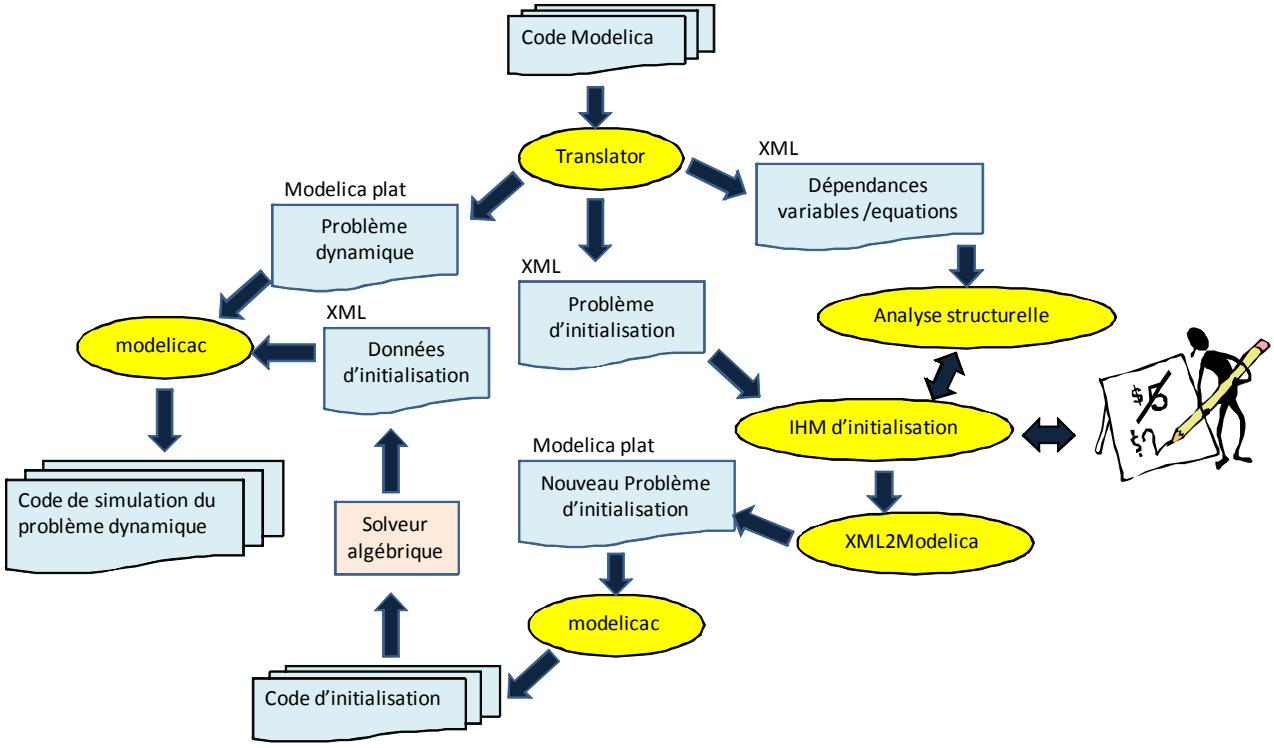

Figure 4 *- Schéma d'aide à l'initialisation dans Scicos*

Pour la résolution du problème d'initialisation la partie amont du compilateur (translator) fournit en plus de la description du problème dynamique deux autres sorties :

- un fichier XML contenant la description du problème d'initialisation : ce fichier sert de base aux interfaces graphique d'aide à l'initialisation,
- un fichier XML contenant la description des relations de dépendances entre variables (et paramètres) et équations : ce fichier est utilisée pour la détection des systèmes sur-contraints et sous-contraints (étape d'analyse structurelle dans le schéma de la figure 4).

L'IHM d'initialisation utilise la description du problème d'initialisation fournie par « translator » pour afficher une interface graphique à l'utilisateur, en lui offrant la possibilité d'intervenir sur :

- le choix des valeurs initiales des variables (et dérivées) et des paramètres,
- le choix des variables (et dérivées) et des paramètres dont les valeurs sont fixées (les valeurs des autres variables et paramètres sont calculées),

- le choix de la méthode de résolution du problème d'initialisation.

Une analyse structurelle est faite sur les choix précédents en utilisant les dépendances entre variables et équations (générées par « translator »). Cette analyse est faites pour déterminer si le système n'est pas réalisable (i.e. n'a pas de solution), en utilisant un algorithme de recherche de couplage dans un graphe biparti.

Après l'analyse précédente, si le système obtenu en prenant en compte les choix de l'utilisateur est structurellement réalisable, ce dernier est converti en code Modelica à plat, puis transmis à la partie avale du compilateur (modelicac) pour simplification et génération de code « C ». Un solveur algébrique est ensuite appliquée au code « C » généré pour calculer les valeurs initiales des variables, variables dérivées, et paramètres. Ces résultats (données d'initialisation dans le schéma de la figure 4), ainsi que la description du problème dynamique sont utilisés conjointement par « modelicac » pour générer le code final pour la simulation dynamique.

1.7.3. Aide à l'initialisation dans AMESim

Aucun développement prévu dans le cadre de SimPA2.

1.7.4. Liste des travaux

Fonctionnalité	Responsable	Produit
Génération de la description du problème d'initialisation	LMS-Imagine	Compilateur Modelica
Génération de la description des relations de dépendances variables/équations avant simplification	LMS-Imagine	Compilateur Modelica
Module de conversion de XML vers Modelica	LMS-Imagine	Compilateur Modelica
Analyse structurelle pour l'aide à l'initialisation	INRIA	Scicos
IHM d'initialisation	INRIA	Scicos
Publication d'articles sur l'initialisation	INRIA	Scicos, Compilateur Modelica

1.8. Accès aux informations sur la structure des équations, sur leur orientation et sur les conditions d'inversibilité du modèle sous l'ensemble des formalismes

1.8.1. Besoin fonctionnel

Offrir la possibilité de visualiser pour un modèle physique donné les informations sur la structure des équations, leurs orientations, ainsi que les conditions d'inversibilité du modèle.

1.8.2. Accès aux informations structurelles des modèles

Les informations structurelles d'un modèle : structure des équations, leurs orientations, ainsi que les conditions d'inversibilité sont mis à disposition par le module d'analyse structurelle de l'inversibilité (décrit au § 2.1).

1.8.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Accès aux informations structurelles des modèles	Alyotech	Module d'analyse structurelle pour l'inversion de modèles

2. Vérification statique (avant simulation)

2.1. Vérification de l'inversibilité du modèle par analyse structurelle et fourniture des conditions d'inversibilité

2.1.1. Besoin fonctionnel

Fournir une méthodologie pour l'analyse des propriétés d'inversibilité des modèles physiques, permettant de déterminer :

- Si un modèle n'est pas inversible,

- Si un modèle est inversible sous certaines conditions.

Un prototype permettant de mettre en œuvre cette méthodologie doit être développée dans le cadre du projet.

2.1.2. Choix techniques

Suite à l'étude réalisée sur les liens entre les formalismes « Bond graph » et Modelica (voir § 1.4.2), il a été décidé de traiter le formalisme « Bond graph » en amont de la chaîne de compilation Modelica. L'idée est d'analyser l'inversibilité d'un modèle en se basant sur les propriétés structurelles de son modèle « Bond graph » : lignes de puissance, causalité des phénomènes, ordres du modèle, chemin causaux, ordres de ces chemins, ...

2.1.3. Analyse structurelle pour l'inversion de modèles physiques

Une spécification détaillée est réalisée par le laboratoire AMPERE de l'INSA de Lyon [9]. Ce document sert de base pour la conception d'un module permettant d'analyser un modèle « Bond Graph » et de fournir les résultats de l'analyse en termes de conditions sur la structure du modèle (voir [9] pour les détails des algorithmes mis en œuvre). Ce module interagit avec l'outil « Bond Graph » MS1 [10], en lui fournissant les conditions à prendre en compte pour générer une représentation inversible du modèle décrite en Modelica. Le code Modelica généré par MS1 (voir figure 5) est ensuite fourni au compilateur Modelica pour l'analyse et la génération de code à destination du simulateur.

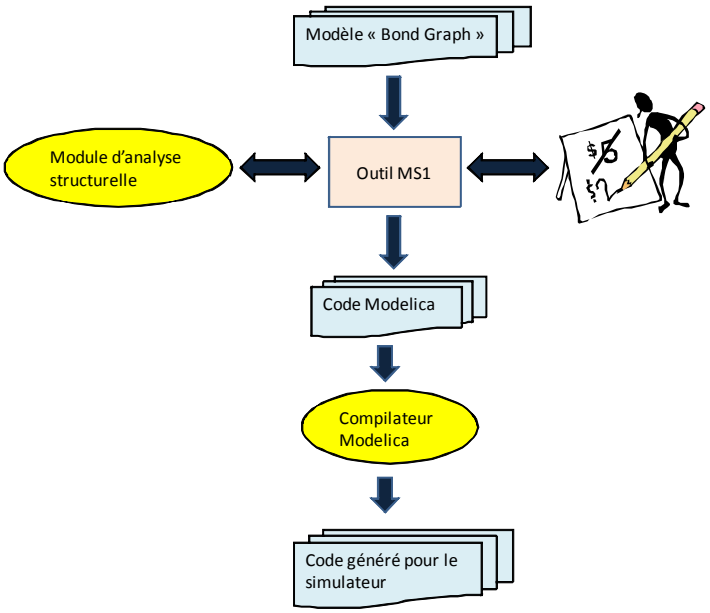

Figure 5 *– Utilisation du module d'analyse structurelle pour l'inversion de modèles*

2.1.4. Liste des travaux

Fonctionnalité	Responsable	Produit
Spécification d'une méthodologie pour l'analyse structurelle de l'inversibilité des modèles physiques.	INSA-Lyon	Module d'analyse structurelle pour l'inversion de modèles
Développement du module d'analyse structurelle pour l'inversion de modèles.	Alyotech	Module d'analyse structurelle pour l'inversion de modèles

2.2. Détection des problèmes mal posés

2.2.1. Besoin fonctionnel

Détecter au plus tôt (lors de la compilation) les modèles erronés, et fournir suffisamment d'informations aux utilisateurs pour localiser et corriger ces erreurs.

2.2.2. Gestion des messages d'erreurs dans le compilateur

Contrairement au compilateur de SimPA qui ne faisait aucun diagnostic des erreurs rencontrées lors de la compilation du code Modelica, le nouveau compilateur affiche des messages d'erreurs clairs, avec des informations précises sur les positions des erreurs rencontrées. Une liste exhaustive avec une description détaillée des erreurs qui peuvent être rencontrées lors de la compilation est fournie avec la documentation du produit SimPA2.

2.2.3. Analyse structurelle de la solvabilité dans le compilateur

La partie avale du compilateur effectue une analyse du système d'équations à résoudre pour détecter au plus tôt si le système n'admet pas de solution, en se basant sur le graphe des dépendances entre variables et équations. Cependant cette analyse est globale et ne permet pas d'identifier la partie du système qui est à l'origine du problème.

2.2.4. Liste des travaux

Fonctionnalité	Responsable	Produit
Gestion des messages d'erreurs	LMS-Imagine	Compilateur Modelica

2.3. Détection des index élevés

2.3.1. Besoin fonctionnel

Les problèmes à états liés (et en particulier ceux à index élevé) sont très difficile à résoudre, et dans certains cas ce sont les symptômes d'erreurs de modélisation, il est donc important d'avertir l'utilisateur lorsque de telles situations se produisent.

2.3.2. Détection et résolution des systèmes à états liés

Les systèmes d'index élevé font partie de l'ensemble plus général des systèmes à états liés, i.e. tout système d'index élevé est un système à états liés, mais pas l'inverse. Les systèmes à états liés sont difficiles à initialiser, puisqu'ils contiennent des relations algébriques cachées entre variables d'état. Un développement a été réalisé pour détecter si un système est à état lié, et tenter de réduire le nombre d'état si possible. Seuls les systèmes avec des relations linéaires entre variables d'état sont traités. Le compilateur peut effectuer plusieurs tentative pour réduire le nombre d'état d'un tel système. Voici un exemple de messages affichés par le compilateur pour un système avec deux variables d'état lié :

```
Trying to reduce state... Failed.
Trying to reduce state... One state variable removed.
Trying to reduce state... Failed.
```

2.3.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Détection et résolution des systèmes à états liés	LMS-Imagine	Compilateur Modelica

2.4. Vérification du réseau (cohérence des ports/connexions)

2.4.1. Besoin fonctionnel

Vérifier la cohérence des connexions au cours de la phase de compilation du code Modelica.

2.4.2. Développements dans le compilateur Modelica

Le compilateur de SimPA est amélioré en ajoutant :

- une vérification plus rigoureuse des types des variables à connecter : propriété « flow », type de base, dimensions pour les tableaux…,
- une vérification des contraintes de connexion : « input » et « output »,
- utilisation des sous-connecteurs,
- utilisation des connecteurs contenus dans des sous-modèles (modèles hiérarchiques),
- utilisation du mot clé « connector ».

2.4.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Amélioration du support des connecteurs	LMS-Imagine	Compilateur Modelica

2.5. Détection des Boucles Algébriques

2.5.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de détecter les boucles algébriques dans les deux types de modèles standards et Modelica.

2.5.2. Détection des boucles algébriques dans AMESim

AMESim détecte les boucles algébriques entres les sous-modèles standards et importés (à partir d'un code Modelica), en utilisant une approche en boîte noire. AMESim ne voit pas les relations existantes entre entrées et sorties d'un même sous modèle, et considère que toutes les sorties dépendent de toutes les entrées. Une telle approche peut détecter plus de boucles algébriques qu'il n'en existe réellement.

2.5.3. Détection des boucles algébriques dans Scicos

La détection des boucles algébriques dans Scicos est plus fine que celle d'AMESim, puisqu'il offre la possibilité d'ajouter les informations de dépendances entre entrées et sorties des sous modèles. Pour les modèles Modelica cette information peut être automatiquement générée par le compilateur ce qui permet à Scicos de détecter de manière plus précise l'existence de boucles algébriques.

2.3.4. Liste des travaux

Aucun développement prévu.

3. Génération de modèle

3.1. Informations sur le système

3.1.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de visualiser les propriétés des modèles compilés à partir d'un code Modelica : nombre d'équations après simplification, liste des variables d'états, liste des paramètres, liste des variables des ports…

3.1.2. Génération des informations sur le modèle dans AMESim

Le compilateur Modelica pour AMESim génère en plus du code « C », un fichier de spécification (écrit en XML) décrivant toutes les propriétés du modèle obtenu après simplification : la liste des variables de port, variables d'état, paramètres… Ces informations sont utilisées par l'IHM d'AMESim et mises à disposition de l'utilisateur. De plus l'assistant d'import Modelica affiche plusieurs autres informations : nombre de variables avant et après simplification, nombre d'équations…

3.1.3. Génération des informations sur le modèle dans Scicos

Le compilateur Modelica pour Scicos génère en plus du code « C » :

- Un fichier XML contenant la description des relations de dépendances variables/équations dans le système obtenu après simplification,
- Une description de la matrice Jacobienne sous forme analytique.

Comme dans AMESim, l'IHM de Scicos met aussi à la disposition de l'utilisateur plusieurs informations : nombre de variables avant et après simplification, nombre d'équations, nombre d'entrées et de sorties…

3.1.4. Liste des travaux

Fonctionnalité	Responsable	Produit
Génération de la description des relations de dépendances variables/équations après simplification	LMS-Imagine	Compilateur Modelica pour Scicos

3.2. Génération de code temps réel embarqué à partir d'un contrôleur discret

3.1.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de générer du code temps réel embarqué à partir d'un contrôleur discret.

3.1.4. Liste des travaux

Aucun développement prévu.

4. Initialisation

4.1. Calcul Inverse

4.1.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de définir et résoudre l'état initial comme un problème inverse.

4.1.2. Résolution de l'état initial comme un problème inverse dans Scicos

Cette fonctionnalité est entièrement supportée dans Scicos (voir le schéma de la figure 4). L'utilisateur a la possibilité de définir l'état initial en utilisant :

- les sections « initial equation » dans le code Modelica,
- les attributs « start » et « fixed » pour définir la valeur initiale des variables,
- les équations de déclaration pour définir les valeurs initiales des paramètres.

Le compilateur Modelica génère une description XML du problème d'initialisation avec :

- l'introduction de variables algébriques pour les dérivées (pour déterminer les valeurs initiales des dérivées de variables), ce qui nécessite de dériver formellement les expressions et identifier les variables effectivement dérivées,
- la prise en compte des sections « initial equation »,
- la prise en compte des équations de la partie dynamique (qui doivent aussi être vérifiées à l'initialisation).

Cette description XML est utilisée pour résoudre le problème d'initialisation (voir la description de l'aide à l'initialisation dans Scicos § 1.7.2). La résolution du problème d'initialisation passe par la génération d'une description en Modelica (à plat) du problème initial, puis la génération de code pour Scicos, en utilisant « modelicac » (cette étape nécessite une adaptation de la génération de code).

Le code généré est ensuite utilisé par Scicos pour résoudre le problème d'initialisation avec l'une des méthodes suivantes (choisie par l'utilisateur via l'IHM d'initialisation) :

- **Sundials** : qui utilise la méthode Newton modifiée.
- **Fsolve** : qui est le solveur interne de Scilab et utilise la méthode Newton avec une factorisation QR
- **Optim** : qui est un optimisateur interne de Scilab. ce solveur est notamment utilisé pour sortir des zones où la matrice Jacobian est singulière.
- **Homotopy** : qui utilise le « package » HOMPACK (http://www.netlib.org/hompack/) permettant la résolution d'équations par la méthode d'homotopie (cf. [8]).
- **Fsolve** : qui utilise aussi la méthode d'homotopie, mais avec le solveur Fsolve.

Un fichier XML contenant les données d'initialisation est ensuite généré par Scicos, celui-ci est transmis à « modelicac » avec la description du problème dynamique pour la génération du code final de simulation dynamique (voir la figure 4).

4.1.3. Résolution de l'état initial comme un problème inverse dans AMESim

AMESim ne supporte pas cette fonctionnalité pour les raisons suivantes :

- l'effort de développement nécessaire est trop important, par rapport aux ressources disponibles,
- l'utilisation des équations initiales en Modelica pose plusieurs problèmes toujours en cours d'étude :

- systèmes initiaux sur et sous-contraints,
- équations initiales mal définies (incohérence avec la partie dynamique),
- duplication des initialisations (utilisation simultanée de l'attribut « start » et des équations initiales),
- manque de connaissances sur le système qui sera obtenu après l'étape de simplification (les variables d'état ne sont pas connues…), ce qui rend la définition des équations initiales très difficile.

4.1.4. Liste des travaux

Fonctionnalité	Responsable	Produit
Support de la section « initial equation »	LMS-Imagine	Compilateur Modelica
Prise en compte des données d'initialisation à la génération du code pour le problème dynamique	LMS-Imagine	Compilateur Modelica
Adaptation de la génération de code pour la résolution du problème d'initialisation	LMS-Imagine	Compilateur Modelica
Intégration de nouvelles méthodes pour la résolution du problème d'initialisation	INRIA	Scicos

4.2. Résolution de l'état initial comme état stationnaire

4.2.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de résoudre l'état initial comme état stationnaire.

4.2.2. Résolution de l'état initial comme état stationnaire dans AMESim

Cette fonctionnalité est implantée sous AMESim depuis la sortie de la Rev7 (mai 2007). Des travaux pour améliorer les calculs d'initialisation dans AMESim sont planifiés en 2008 (amélioration de l'existant et exploration de nouvelles méthodes).

4.2.3. Résolution de l'état initial comme état stationnaire dans AMESim

Cette fonctionnalité existe sous Scicos depuis le projet SimPA. L'interface graphique d'initialisation permet d'initialiser les modèles en état stationnaire, c'est-à-dire avec les dérivées nulles.

3.2.4. Liste des travaux

Aucun développement prévu.

4.3. Résolution du problème de commutation de modèles pour l'initialisation

4.3.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de définir (en Modelica) dans la section « initial equation » des équations conditionnelles, de la forme :

```
if condition1 then equation1 elseif condition2 then equation 2 else equation 3 end if;
```

Cette fonctionnalité est importante pour résoudre le problème de détermination du bon mode au démarrage d'une simulation.

4.3.2. Développements dans le compilateur Modelica

La syntaxe supportée par le compilateur Modelica a été étendue pour pouvoir utiliser les « équations conditionnelles » de la forme :

```
if condition1 then equation1 elseif condition2 then equation 2 else equation 3 end if;
```

Ces équations sont supportés dans les deux sections « equation » et « initial equation » du code Modelica. Les équations conditionnelles avec des conditions variables au cours de la simulation ne sont pas supportées dans le cadre de SimPA2. En d'autres termes, les conditions (e.g. `condition1`, `condition2` et `condition3` dans l'exemple précédent) doivent dépendre uniquement de constantes ou de paramètres.

4.3.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Support des équations conditionnelles	LMS-Imagine	Compilateur Modelica

5. Simulation

5.1. Calcul direct

5.1.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de résoudre un problème de simulation directe décrit en utilisant le langage Modelica.

5.1.2. Simulation directe sous AMESim

Cette fonctionnalité est supportée par AMESim, voir § 1.2.2.

5.1.3. Simulation directe sous Scicos

Cette fonctionnalité est supportée par Scicos, voir § 1.2.3.

5.1.4. Liste des travaux

Aucun développement supplémentaire prévu.

5.2. Résolution du problème de commutation de modèles pour la simulation dynamique

5.2.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de définir (en Modelica) dans la section « equation » des équations conditionnelles, de la forme :

```
if condition1 then equation1 elseif condition2 then equation 2 else equation 3 end if;
```

Cette fonctionnalité est importante pour décrire des modèles pouvant présenter suivant leur point de fonctionnement des structures d'équations différentes.

5.2.2. Développements dans le compilateur Modelica

Voir § 4.3.2.

3.2.3. Liste des travaux

Aucun développement supplémentaire prévu.

5.3. Traitement des boucles Algébriques (DAE)

5.3.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de :

- résoudre les boucles algébriques à l'intérieur d'un modèle Modelica (au cours de la compilation),
- résoudre les boucles algébriques entre des modèles compilés (Modelica ou non).

5.3.2. Traitement des boucles algébriques dans AMESim et Scicos

Le compilateur SimPA2 détecte et résout les boucles algébriques à l'intérieur d'un modèle Modelica. De plus, AMESim et Scicos sont capables de détecter (voir § 2.5.2 et 2.5.3) et résoudre les boucles algébriques apparaissant entre des sous-modèles (standards ou importés à partir d'un code Modelica).

5.3.3. Liste des travaux

Aucun développement supplémentaire prévu.

5.4. Traitement des systèmes Raides (DAE et ODE)

5.4.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de :
- résoudre les boucles algébriques à l'intérieur d'un modèle Modelica (au cours de la compilation),
- résoudre les boucles algébriques entre des modèles compilés (Modelica ou non).

5.4.2. Traitement des systèmes raides dans AMESim et Scicos

AMESim et Scicos disposent tous les deux de méthodes numériques permettant de traiter les systèmes raides (méthodes BDF adaptées).

5.4.3. Liste des travaux

Aucun développement supplémentaire prévu.

5.5. Traitement des systèmes hybrides

5.5.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de définir et simuler des modèles hybrides (coexistence des aspects discrets et continus).

5.5.2. Spécification du nouveau modèle hybride

La spécification du langage Modelica concernant les aspects hybrides reste très ambiguë (voir le retour d'expérience et bilan SimPA [1]), un travail important d'étude et de recherche a été réalisé par l'INRIA en collaboration avec LMS-Imagine afin de proposer une approche permettant de décrire et traiter les modèles hybrides en utilisant le langage Modelica. Une proposition pour un nouveau modèle hybride inspirée de Scicos a été présentée [11] à la sixième conférence Modelica (Mars 2008). Ce travail est un des principaux apports du projet SimPA2.

5.5.3. Développements dans le compilateur Modelica

Le développement du nouveau modèle hybride proposé par l'INRIA et LMS-Imagine n'est pas prévu dans le cadre de SimPA2, pour les raisons suivantes :
- La spécification d'un modèle hybride pour Modelica est une tâche très couteuse, étant donné le manque de sémantique claire du langage Modelica, et la complexité des constructions langage.
- Les partenaires industriels du projet SimPA2 ont décidé de mettre la priorité sur les développements pour l'initialisation et l'inversion de modèles.
- Il ne suffit pas proposer un modèle hybride pour Modelica, il faut aussi convaincre les membres de l'association Modelica d'opter pour ce choix.
- Le nouveau modèle hybride nécessite une refonte complète du noyau de manipulations symboliques, et une refonte importante de l'analyse sémantique des constructions liées aux aspects discrets.

Pour ces raisons, le développement du nouveau modèle hybride est prévu en dehors du cadre du projet simPA2.

Les aspects discrets supportés par le compilateur SimPA2 sont :
- les réinitialisations d'états continus,
- les états discrets indépendants de type Real (« indépendants » signifie qu'on ne gère pas les cascades d'événements provoquées par des «when» dépendants).

Les principales améliorations du compilateur SimPA2 par rapport à SimPA sont :
- refonte complète du code de gestion des traversées de surface, pour lever les restrictions sur les expressions supportées comme conditions dans les « if » et les « when »,
- le support de l'opérateur « pre ».

5.5.4. Liste des travaux

Fonctionnalité	Responsable	Produit
Spécification des algorithmes pour le traitement des systèmes hybrides	INRIA/LMS-Imagine	Compilateur Modelica
Publication d'articles sur le nouveau modèle hybride	INRIA/LMS-Imagine	Compilateur Modelica
Refonte du code de gestion des traversées de surface	LMS-Imagine	Compilateur Modelica
Support de l'opérateur « pre »	LMS-Imagine	Compilateur Modelica

5.6. Calcul inverse automatisé en cours de simulation

5.6.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de définir et résoudre les problèmes d'inversion dynamique de modèles.

5.6.2. Résolution du problème inverse dynamique

Non supporté par AMESim et Scicos.

5.6.3. Liste des travaux

Aucun développement supplémentaire prévu.

5.7. Couplage de modèles 1D et 3D

5.7.1. Besoin fonctionnel

Offrir dans AMESim et Scicos la possibilité de définir et résoudre les problèmes de couplage 1D et 3D en utilisant Modelica.

5.7.2. Résolution des problèmes de couplage 1D et 3D en utilisant Modelica

Non supporté par AMESim et Scicos.

5.7.3. Liste des travaux

Aucun développement supplémentaire prévu.

6. Outils de diagnostic

6.1. Informations sur la structure des équations effectivement résolues

6.1.1. Besoin fonctionnel

Offrir dans AMESim et Scicos un accès aux informations sur la structure des équations effectivement résolues (après manipulations symboliques) données sous forme littérale compréhensible par l'utilisateur (liste des systèmes d'équations après réduction, …).

6.1.2. Développements dans AMESim et Scicos

Cette fonctionnalité est non supportée par AMESim et Scicos.

6.1.3. Liste des travaux

Aucun développement supplémentaire prévu.

6.2. Accès aux informations sur la structure des équations, leur orientation et les conditions d'inversibilité du modèle (version textuelle)

6.2.1. Besoin fonctionnel

Offrir dans le module d'analyse structurelle pour l'inversion de modèles, un accès aux informations sur la structure des équations, leurs orientations et les conditions d'inversibilité sous une forme textuelle.

6.2.2. Développements dans le module d'analyse structurelle pour l'inversion de modèles

Cette fonctionnalité n'est pas supportée.

6.2.3. Liste des travaux

Aucun développement supplémentaire prévu.

6.3. Messages d'erreurs en cours de simulation

6.3.1. Besoin fonctionnel

Lorsqu'une erreur se produit en cours de simulation, le simulateur (AMESim ou Scicos) doit afficher des messages d'erreurs claires.

6.3.2. Affichage des messages d'erreurs en cours de simulation dans AMESim

AMESim supporte depuis sa création un ensemble de fonctionnalités permettant d'afficher des messages d'erreurs clairs en cours de simulation et d'analyser le système résolu numériquement.

6.3.3. Affichage des messages d'erreurs en cours de simulation dans Scicos

Scicos supporte entièrement cette fonctionnalité.

6.3.4. Liste des travaux

Aucun développement supplémentaire prévu.

7. Environnement informatique

7.1. Plateformes informatiques supportés

7.1.1. Besoin fonctionnel

Les composants informatiques développés dans le cadre de SimPA2 doivent tous fonctionner sous les deux plateformes : Windows, Linux.

7.1.2. Environnements informatiques supportés

Les environnements Windows et Linux sont tous les deux supportés par tous les composants développés dans le cadre de SimPA2.

7.1.3. Liste des travaux

Aucun développement supplémentaire nécessaire.

7.2. Mise à disposition de la documentation utilisateur

7.2.1. Besoin fonctionnel

Mettre à disposition une documentation utilisateur pour chaque composant développé dans le cadre du projet.

7.2.2. Liste des travaux

Aucun développement supplémentaire nécessaire.

8. Les performances

8.1. Besoin fonctionnel

L'utilisation du compilateur Modelica doit permettre d'aborder les « gros modèles » de taille industrielle, et d'améliorer significativement les temps de simulation. Les méthodes de simplification formelle appliquées par le compilateur doivent permettre de réduire considérablement la taille des systèmes effectivement résolus par le solveur et diminuer ainsi les temps de calcul.

8.2. Développements dans le compilateur Modelica pour AMESim

Pour améliorer l'efficacité et la robustesse de la simulation des modèles Modelica importés dans AMESim, un développement a été réalisé pour générer des systèmes explicites, pour lesquelles il existe des méthodes de résolution numérique plus performantes et robustes.

8.2. Liste des travaux

Fonctionnalité	Responsable	Produit
Détection et génération de systèmes explicites.	LMS-Imagine	Compilateur Modelica pour AMESim

9. La portabilité des modèles entre AMESim et Scicos

9.1. Besoin fonctionnel

Assurer la compatibilité des modèles entre les plateformes d'AMESim et Scicos.

9.2. Compatibilité des modèles entre AMESim et Scicos

Dans un premier temps, AMESim et Scicos utilisent le même compilateur Modelica, ce qui garantit que les librairies Modelica écrites pour Scicos, peuvent être directement utilisées (sans aucune modification) sous AMESim et vice versa.

Par la suite, le compilateur Modelica sera amélioré (dans AMESim) pour traiter d'autres classes de problèmes étendues (réduction d'index, algorithmes, fonctions externes avec discontinuités et états internes, partitionnement…) dans le cadre d'une version commerciale du compilateur. Dans cette dernière version, la portabilité de Scicos vers AMESim sera maintenue, mais pas l'inverse. C'est-à-dire que certains modèles écrits sous AMESim pourront ne pas fonctionner sous Scicos.

9.2. Liste des travaux

Aucun développement supplémentaire prévu.

10. La documentation

10.1. Besoin fonctionnel

Livrer une documentation complète pour :
- le compilateur Modelica,
- l'utilisation de Modelica dans AMESim,
- l'utilisation de Modelica dans Scicos,
- l'utilisation du module d'analyse structurelle pour l'inversion de modèles.

10.2. Documentation du compilateur Modelica

La documentation fournie avec le compilateur Modelica comportera la description :
- des fonctionnalités du compilateur
- des erreurs syntaxiques et sémantiques
- des éléments non supportés du langage Modelica
- de la procédure d'installation du compilateur.

10.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Documentation du compilateur Modelica	LMS-Imagine	Compilateur Modelica
Documentation de Modelica dans AMESim	LMS-Imagine	AMESim

Documentation de Modelica dans Scicos	INRIA	Scicos
Documentation du module d'analyse structurelle pour l'inversion de modèles	Alyotech	Module d'analyse structurelle pour l'inversion de modèles

11. Autres besoins fonctionnels

11.1. Besoins liés au retour d'expérience sur SimPA

11.1.1. Besoin fonctionnel

L'un des résultats du retour d'expérience et bilan sur le projet SimPA est la nécessité de supporter l'héritage, qui est souligné comme un élément important pour la modélisation des actionneurs (c.f. [1]), car permettant de gérer plus efficacement les variantes de modélisation.

11.1.2. Développements dans le compilateur Modelica

Les concepts d'héritage et de modèles partiels sont ajoutés au compilateur, afin d'offrir aux utilisateurs plus de facilité dans l'écriture, la réutilisation et le partage du code Modelica.

11.1.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Support de l'héritage	LMS-Imagine	Compilateur Modelica

11.2. Besoins liés aux premiers tests sur SimPA2

11.2.1. Besoin fonctionnel

Les premiers tests réalisés sur le premier prototype du compilateur SimPA2 par les partenaires industriels (c.f. tests de PSA), ont montrés la nécessité de d'améliorer le compilateur pour supporter :

- les conversions implicites (entiers vers réels), pour simplifier les écritures,
- les fonctions et opérateurs scalaires (acos, asin…), ceux-ci ne sont pas tous supportés et il est nécessaire d'écrire par exemple « Modelica.math.sin » pour appeler la fonction « sin ».

11.2.2. Développements dans le compilateur Modelica

Le compilateur Modelica est amélioré pour supporter :

- les fonctions et opérateurs scalaires non supportés par l'ancien compilateur : acos, acosh, asin, asinh, atan, atanh, cosh, sinh, pre, div, mod, rem, ceil, floor,
- la possibilité d'appeler les fonctions prédéfinies sans les faire précéder par l'expression « Modelica.Math »,
- les conversions implicites (entiers vers réels), dans toutes les formes d'expressions et constructions du langage Modelica.

11.2.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Amélioration du support des fonctions et opérateurs scalaires	LMS-Imagine	Compilateur Modelica
Support des conversions implicites entre entiers et réels	LMS-Imagine	Compilateur Modelica

11.3. Besoins liés au développement de librairies

11.3.1. Besoin fonctionnel

Pour faciliter le développement de librairies Modelica il est nécessaire de supporter l'utilisation des paquetages en Modelica.

11.3.2. Développement dans le compilateur Modelica

Contrairement au compilateur de SimPA, le nouveau compilateur (SimPA2) supporte les classes de type « package », ce qui permet de structurer et grouper les classes Modelica dans une hiérarchie de paquetages et sous-paquetages, pour une plus grande lisibilité, facilité d'utilisation et partage du code source. L'encapsulation des définitions est aussi supportée grâce à l'utilisation des éléments « protected », ce qui permet de rendre certaines informations accessibles uniquement à l'intérieur d'un même package. La propriété « encapsulated » a aussi été introduite pour rendre un paquetage complètement indépendant des autres, et faciliter ainsi le déplacement et l'utilisation des informations qu'il contient. L'utilisation des paquetages offre aussi d'autres facilités, comme l'élimination des conflits de noms et la réduction de la complexité du code.

11.3.3. Liste des travaux

Fonctionnalité	Responsable	Produit
Support des paquetages	LMS-Imagine	Compilateur Modelica

Chapitre III : Limitations et récapitulatif des réalisations

1. Principales limitations du compilateur Modelica

Case vide : non supporté, L : supporté avec des limitations, X : complètement supporté			
Fonctionnalité	Compilateur SimPA	Compilateur SimPA2	Version commerciale mai 2008
Fonctions et opérateurs prédéfinis	L	L	L
Types prédéfinis	L	L	L
Utilisation de packages, records, blocks et connecteurs		X	X
Héritage		X	X
Tableaux statiques		X	X
Tableaux dynamiques			
Equations conditionnelles		X	X
Composants remplaçables			
Equations initiales		X	
Fonctions externes	L	L	L
Algorithmes			
Gestion des messages d'erreurs		L	L
Annotations		L	L
Support des aspects discrets		L	L

Pour le compilateur SimPA2, les limitations (indiquées par « L » dans le tableau ci dessus) sont :

- Certains opérateurs prédéfinis ne sont pas supportés : assert, terminate, initial, terminal, change, edge,
- Les opérations sur les tableaux : multiplication de matrices, division par des scalaires, addition et soustraction et équations vectorielles sont toutes supportés, à l'exception de certains opérateurs built-in comme « transpose », « skew » et « symmetric ».
- Le type « Real » est entièrement supporté,
- Les types « Integer » et « String » ne sont supportés que pour la définition de paramètres et constantes,
- Les types « Boolean » et « enumeration » ne sont pas supportés,
- Les fonctions externes supportées sont celles :
  - ayant au maximum un seul « output » de type « Real »,
  - pour lesquelles tous les « inputs » sont des tableaux de « Real » ou des scalaires de type « String », « Integer » ou « Real »,
  - qui sont pures et sans aucune discontinuité.
- L'affichage des messages d'erreurs est géré et documenté seulement pour la partie amont du compilateur,
- Les erreurs syntaxiques sur les annotations sont bien supportées par le compilateur, mais aucune information sur les annotations n'est générée. C'est à l'outil de simulation de les récupérer et les interpréter,
- Concernant les aspects discrets, les éléments supportés sont :
  - les réinitialisations d'états continus,
  - les états discrets indépendants de type Real (« indépendants » signifie qu'on ne gère pas les cascades d'événements provoquées par des «when» dépendants).

2. Tableau récapitulatif des fournitures du projet

Certains des projets liés à l'intégration de Modelica dans les logiciels AMESim et Scicos sont livrés en trois étapes:

- version prototype : qui est une première version, destinée aux tests en interne,
- version projet : qui est réservé à deux ou trois clients pour tester le produit et avoir les premiers retours des utilisateurs,
- version industrielle : qui est la version finale destinée à tous les clients.

Les versions du compilateur Modelica sont numérotées comme suit : XXX/YYY, avec XXX la version de la partie amont et YYY la version de la partie avale.

Date de livraison	Titre	Version (amont/avale)	Responsable	Description
06/2007	Première version du compilateur	1.0/1.10.0	LMS Imagine	- Support de l'attribut « fixed », - Support des paquetages - Support de l'héritage, - Amélioration du support des connecteurs, - Support des conversions implicites, - Gestion des messages d'erreurs
	Assistant d'import Modelica pour AMESim	AMESim Rev7A	LMS Imagine	- Importation de code Modelica dans l'environnement AMESim, - Simulation directe des modèles. - Mise à disposition de la documentation.
	Intégration dans Scicos	Scicos 4.1	INRIA	- Intégration du compilateur dans Scicos, - Intégration de Sundials dans Scicos, - Editeur de code Modelica. - Mise à disposition de la documentation.
12/2007	Spécification détaillée du produit SimPA2	1.6	LMS Imagine	- Première version de la spécification détaillée SimPA2.
	Module de conversion XML vers Modelica	1.0	LMS Imagine	- Conversion de XML vers Modelica
02/2008	Version intermédiaire du compilateur	1.1/1.10.1	LMS Imagine	- Tableaux de dimensions paramétriques, - Tableaux définis en compréhension, - Support des équations conditionnelles, - Amélioration du support des fonctions et opérateurs scalaires. - Refonte du code de gestion de traversée de surface, - Détection et génération de systèmes explicites pour la cible AMESim.
03/2008	Publication à la conférence Modelica 2008		INRIA, LMS Imagine	- Proposition d'un nouveau modèle hybride, - Initialisation de modèles Modelica dans Scicos
05/2008	Assemblage Modelica sous AMESim (version prototype)	AMESim Rev 8A	LMS Imagine	
	Aide à l'initialisation sous Scicos (version prototype)	Scicos 4.2	INRIA	- IHM d'initialisation - Intégration de nouvelles méthodes pour la résolution du problème d'initialisation - Analyse structurelle pour l'aide à l'initialisation.
06/2008	Spécification d'une méthodologie pour l'analyse structurelle de l'inversibilité des modèles physiques.		INSA-Lyon	- Document de spécification
08/2008	Version finale du compilateur	1.2/1.11.0	LMS Imagine	- Opérations sur les tableaux, - Support des fonctions externes,

				- Support des paramètres et constantes de type « String » et « Integer », - Génération de la description des relations de dépendances variables/équations avant et après simplification, - Support de la section « initial equation », - Génération de la description du problème d'initialisation, - Prise en compte des données d'initialisation à la génération du code pour le problème dynamique, - Adaptation de la génération de code pour la résolution du problème d'initialisation, - Support de l'opérateur « pre », - Amélioration de la visualisation des valeurs des paramètres et variables dans AMESim.
08/2008	Spécification détaillée du produit SimPA2	1.8		
12/2008	Assemblage Modelica sous AMESim (version projet)	AMESim Rev 8B	LMS Imagine	
	Aide à l'initialisation dans Scicos (version projet)	Scicos 4.3	INRIA	
	Module d'analyse structurelle pour l'inversion de modèles	1.0	Alyotech	- Développement du module, - Intégration du module à MS1, - Mise à disposition de la documentation, - Accès aux informations structurelles des modèles.
02/2009	Documentation du compilateur		LMS Imagine	

Références

[1] Projet SimPA2 - Expression de besoins pour un environnement de modélisation et simulation basé sur un compilateur Modelica open source, aout 2006.

[2] RNTL – Appel à propositions 2005, Annexe technique du projet SimPA2.

[3] Définition du tableau des exigences fonctionnelles pour l'établissement de la roadmap SimPA2, février 2007.

[4] Accord de consortium du projet SimPA2.

[5] Modelica – A unified object-oriented language for physical systems modeling, Language specification version 3.0, Sept. 5, 2007.

[6] S. Furic (2007) Software design specification for SimPA2 – SP2.

[7] P. Fritzson (2004) Principles of object oriented modeling and simulation with Modelica 2.1, Wiley-IEEE Press.

[8] M. Najafi et R.Nikoukhah (2008) Initialization of Modelica models in Scicos, to appear in 6th International Modelica Conference.

[9] W. Marquis-Favre, D. Thomasset, A. Jardin, (2008) Document support pour la spécification et le développement du module d'analyse structurelle du projet SimPA2.

[10] F. Lorenz, (1997), *Modelling System 1*, users manual, Lorenz Simulation, Liege, Belgium.

[11] R. Nikoukhah and S; Furic (2008), Synchronous and asynchronous events in Modelica: Proposal for an improved hybrid model, 6th International Modelica Conference, March 3-4, Bielefeld, Germany.